

Simple Data Compression by Differential Analysis using Bit Reduction and Number System Theory

Debashish Chakraborty¹, Sandipan Bera², Anil Kumar Gupta², Soujit Mondal²

Department of Computer Science & Engineering¹ and Information Technology²

St. Thomas' College of Engineering. & Technology,

Kolkata-23, West Bengal, India

sunnydeba@gmail.com, sandipan.bera@gmail.com, anilgupta00749@gmail.com

Abstract— This is a simple algorithm which is based on number theory system and file differential technique. It employs a technique which unlike other is independent of repetition and frequency of character. In the algorithm original file is broken into some small files using differential techniques and then every small file is considered as certain n-base number system where n is the number of different characters in the file. Now the compression is done converting this n-base number system to binary number system. The main concept behind this algorithm is to save the bits by converting the higher number system to lower one. It is a simple compression and decompression process which is free from time complexity.

Keywords— Number system, Bit saving, Differential technique.

I. INTRODUCTION

Data compression or source coding is the process of encoding information using fewer bits (or other information bearing units) than a non encoded representation would use through specific use of encoding schemes [1,2,3,5]. It follows that the receiver must be aware of the encoding scheme in order to decode the data to its original form. The compression schemes that are designed are basically trade-offs among the degree of data compression, the amount of distortion introduced and the resources (software and hardware) required to compress and decompress data. The primary reason behind doing so is to reduce the storage space required to save the data, or the bandwidth required to transmit it. Although storage technology has developed significantly over the past decade, the same cannot be said for transmission capacity. Data compression schemes may broadly be classified into – 1. Lossless compression and 2. Lossy compression. Lossless compression algorithms usually exploit statistical redundancy in such a way as to represent the sender's data more concisely without error. Lossless compression is possible because most real world data has statistical redundancy. Another kind of compression, called lossy data compression is possible if some loss of fidelity is acceptable. It is important to consider that in case of lossy compression, the original data cannot be reconstructed from the compressed data due to rounding off or removal of some parts of data as a result of redundancies. These types of compression are also widely used in Image compression [10,11,12]. The theoretical background of compression is provided by information theory and by rate

distortion theory [6,7,8,9]. This is a new data compression algorithm. It is not so much dependent about the characters repetition. Though its compression ratio is not so much but the differential technique makes the compression ratio to a constant value. The main advantage of this technique is that it can compress the output file which is produced after applying certain compression techniques on a file. So this algorithm gives good result when it is used for hybridization with other algorithm.

II. ALGORITHM STRATEGY

At first we need to break the file into some parts or some sub files. This differential condition is that the file will be broken when the no. of distinct characters will be n. Here n is an integer number which will be varied file to file. Now the total number of different characters is n then we consider the file as n base number system where each character is one of the elements of this n base number system. So, now the requirement is indexing of the different characters present in the sequential file from 0 to n-1. Now we pick g numbers of characters from the original file at a time and then taking the index number of each character calculates its decimal value according to number theory system. Now represent this decimal value by s numbers of bits in binary format to get the compressed file. Here is a certain relation between g and s which is established by mathematical analysis in the later section. To get the best compression proper selection of this g and s parameter is very necessary.

III. CALCULATION

Let take an n-base number system where n can be any integer. So the value of maximum element of this number system is n-1. Now we take g no. of such element of this number system. Then the maximum value of this g no. of elements would be,

$$\text{Max_val} = (n-1)*n^0 + (n-1)*n^1 + (n-1)*n^2 + \dots + (n-1)*n^{g-1} \\ = (n-1)[(n^g - 1)/(n - 1)] = (n^g - 1).$$

Then, similarly with s no. of bits the maximum binary value is $(2^s - 1)$.

So, now if we transfer this n-base number system to binary number system then,

$$\text{We can say, } (n^g - 1) = (2^s - 1) \Rightarrow n^g = 2^s \Rightarrow g * \log(n) \\ = s * \log(2) \Rightarrow s/g = \log(n)/\log(2) \\ \Rightarrow s/g = 3.322 * \log(n)$$

Now, if we consider a file as a number system where each

character is an element of this system then actually g bytes can be replaced by s bits. So, we can say $8g$ bits will be replaced by s no. of bits. Now, we can say the $\% \text{compression} = (1 - s/(8g)) * 100\%$
 $= (1 - 0.41525 * \log(n)) * 100\%$ where n is the no. of different types of characters in the file.

IV. ALGORITHM

Steps for Compression:

1. Begin
2. Set store[0]=(space)
3. Set count=1
4. Take an empty one order matrix with size n . Let the matrix is store[].
5. Do Until(end of file)
6. Read the file and pick the character from the file.
7. If the character is present in the matrix store[] then go to step 8 else go to step 4 .
8. Set store[count]=character
9. count=count+1
10. If count = n then go to step 11 else go to step 5 .
11. Break the file including the last character which is read.
12. count=1
13. Go to step 5
14. End of Loop.
15. Apply the steps from step 15 to step 27 on each sub file for getting the compression.
16. Find out the different types of characters in the file.
17. Store the different characters in an array. Sequence[]={sorted character set}.
18. Count the no. of different characters. Now it will be n . So here we consider the file as a n -base number system.
19. $g = \text{select an integer } g$.
20. $s = \text{Ceiling value of } (3.322 * g * \log(n))$
21. Do until (end of file)
22. Pick up g no. of characters from the original file at a time.
23. Find the position of the each picked up character within the Sequence[] and store the positional value into another array pos[]. Let pos[]={ $a_0, a_1, \dots, a_{g-2}, a_{g-1}$ }.
24. $\text{pos_val} = a_0 * n^0 + a_1 * n^1 + \dots + a_{g-2} * n^{g-2} + a_{g-1} * n^{g-1}$.
25. Now represent the pos_val with s no. of bits in binary system. Put these bits into a file Result.
26. Go to step 21.
27. End of loop
28. Result file is the compressed file.
29. End

Steps for Decompression:

1. Begin
2. Set $j=0$
3. Sequence[] = {Character set of j^{th} Sub file.}
4. Do until end of compressed string or File.
5. Take s no. of bits at a time and then calculate its decimal value.

6. Let the decimal value is val.
7. Divide val by n , until we store g no. of remainder in an array rem[]. Let rem[]={ $c_0, c_1, c_2, \dots, c_{g-2}, c_{g-1}$ }.
8. Set $i=0$ and Flag=0.
9. While($i < g$ and Flag=0)
10. If rem[i]=Sequence[$n-1$] then Go to step 11 else Go to step 13.
11. Set Flag=1.
12. Go to step 9.
13. $i=i+1$
14. Go to step 9.
15. End of While Loop.
16. If Flag=1 then Go to
17. Now map the each element of the array rem[] with the Sequence[] which is mentioned in Compression routine. Let rem[]={ $c_0, c_1, c_2, \dots, c_{g-2}, c_{g-1}$ }. Then put the Sequence [c_0], sequence[c_1],.....sequence [c_{g-1}] into a file extract.
18. Go to step 4.
19. Now map the 0 to i^{th} element of the array rem[] with the Sequence[] which is mentioned in Compression routine. So, put the Sequence [c_0], Sequence[c_1],..... Sequence[c_i] in the file extract.
20. Set $j=j+1$
21. Sequence[] = {Character set of j^{th} Sub file.}
22. Go to Step 4.
23. End of Loop.
24. Extract will be the decompressed File.
25. End

V. ALGORITHM ILLUSTRATION

For example consider a text like 'Bob is a boy'. Total no of characters are 12 but the character set is [B,o,b,i,s,a,y,]. They can be indexed as {B=0,o=1,b=2,i=3,s=4,a=5,y=6 and 7 for space}. Since number of characters in character set is 8, therefore $n=8$ and take $g=4$. So $s = 3.322 * (\log 8) * 4$. $s = 12.00243$. Taking ceiling value of s we get $s=13$. Now take 4 characters from original string at a time and calculate the decimal value of the index. The first 4 characters {B,o,b, } and the index position are {0,1,2,7} and calculate it in decimal value in such a way, $0*8^0 + 1*8^1 + 2*8^2 + 7*8^3 = 3720$. Again convert 3720 into binary to get a 13 bit value, put in the file in (0111010001000). So by representing in this way the total size of compressed file will be $13*3 = 39$ bits, whereas the original size of the text was $8*12 = 96$ bits. Now in the decryption process, convert the compressed bit string into the decimal value. So converting $(0111010001000)_2 = (3720)_{10}$. Again convert this 3720 with $n=8$ base number system then we get {7,2,1,0} and then reverse it {0,1,2,7} . After mapping with index, we get back the original string {B,o,b, }. In such a way decryption process can be done.

VI. ALGORITHM DISCUSSION

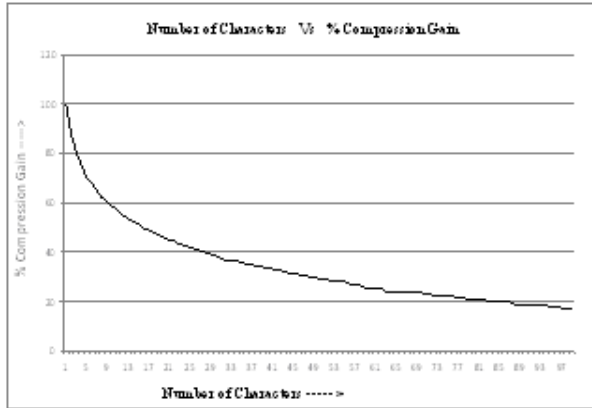


Fig. 1 The graph for No. Of distinct characters in a File Vs Compression gain without differential technique.

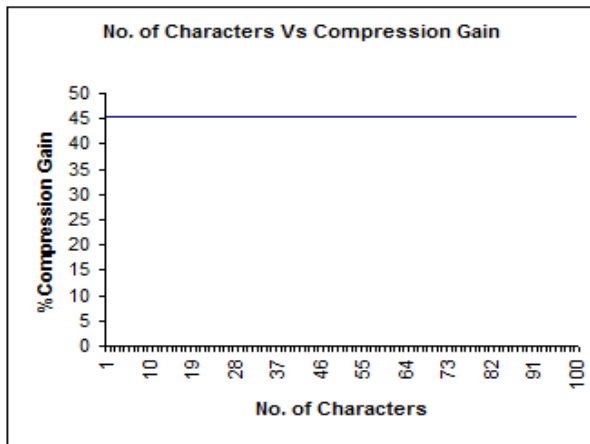


Fig. 2 The graph for No. Of distinct characters in a File Vs Compression gain applying differential technique. (Differential condition is, $n=20$) .

TABLE I

FILE	ORIGINAL SIZE	HUFFMANN (in %)	LZW (in %)	PROPOSED ALGORITHM (in %)
Sample1.txt	2.46KB	36.65%	43.96%	44.71%
Sample2.txt	3.58KB		44.18%	44.71%
Sample3.rtf	2.67KB	40.82%	43.82%	44.56%
Sample4.java	4.12KB	34.95%	44.66%	44.67%
Sample5.c	4.43KB	33.86%	45.89%	44.24%
Sample6.c	9.76KB	39.53%	45.22%	44.56%
Sample7.cpp	5.78KB	37.25%	43.71%	44.38%
Sample8.cpp	3.18KB	39.53%	45.22%	44.56%
Sample9.cpp	5.38KB	36.35%	43.71%	44.48%
Sample10.rtf	8.74KB	35.57%	44.33%	44.76%
Sample11.txt	9.24KB	39.12%	44.27%	44.96%
Sample12.txt	7.29KB	34.13%	45.12%	44.12%

Fig. 3 Compression Gain of proposed algorithm compared with existing algorithms

A. Characters Set Size and Compression gain

The above mentioned graph(Fig.1) just show how the compression gain varies with size of characters set when the differential technique is not applied i.e. no breaking of files. It gives better compression when the size of characters set is lesser. Here we can find some similarity with other well known statistical data compression techniques like Huffman algorithm

B. Differential Technique and Compression gain

The above mentioned table and graph(Fig.2) shows when we apply differential technique then we get constant compression. The selection of no. characters(variable 'n' in the compression routine) on which the breaking of the file is dependent is very important. This number should be the trade of between the compression gain and the no. of file breakings. If the number is very small then the no. of file breakings will be increase and it increase the complexity so much. Though it improves the compression gain also. Similarly if the selected number is so high then the no. of file breakings will be decreased which decrease the complexity also. But on the other hand compression gain will be deteriorated also. So the selected number should be a certain value by which the algorithm performs with a moderate compression gain and complexity.

VII. CONCLUSIONS

In this paper a new data compression algorithm is introduced. The unique feature of this algorithm is its simplicity. An entirely different technique is employed to reduce the size of text files. The technique of 'saving bits' is employed in this algorithm. Since every character is taken care of, so the output codes do not depend upon the repetition, like most of the other compression algorithms. Different combination of characters can be represented by fewer numbers of bits. After the code formation, ASCII replaces the binary numbers, which finally reduces the file size. The compression algorithm takes $O(n)$ time, where n is the total number of characters in the file. Since the differential breaking follows Divide and Conquer policy, it takes $O(n \log n)$ time. So, the total computation time required for this algorithm is proportional to $O(n \log n)$. Quite a lot of research and findings led to the conclusion that there are no such algorithms in data compression that lay emphasis on differential compression based on number theory and bit reduction.

ACKNOWLEDGMENT

First, we would like to thank Professor Subarna Bhattacharjee[4], for her valuable advice, support and constant encouragement. Her constant criticisms and reviews gave us the conceptual clarity. We owe a significant lot to all faculty members of the Department of Computer Science and Engineering and the Department of Information Technology. It was in one such class of Design and Analysis of Algorithms that we first envisioned the above algorithm. We would also

like to thank our friends for patiently enduring our explanations. Their reviews and comments were extremely helpful. And of course, we owe our ability to complete this project to our families whose love and encouragement has been our cornerstone.

REFERENCES

- [1] J.Ziv and A. Lempel, "Compression of individual sequences via variable length coding", IEEE Transaction on Information Theory , Vol 24: pp. 530 – 536, 1978.
- [2] J.Ziv and A. Lempel, "A universal algorithm for sequential data compression", IEEE Transaction on Information Theory , Vol 23: pp. 337 – 343, May 1977.
- [3] Gonzalo Navarro and Mathieu A Raffinot, "General Practical Approach to Pattern Matching over Ziv-Lempel Compressed Text", Proc. CPM'99, LNCS 1645, Pages 14-36.
- [4] S. Bhattacharjee, J. Bhattacharya, U. Raghavendra, D.Saha, P. Pal Chaudhuri, "A VLSI architecture for cellular automata based parallel data compression", IEEE-2006, Bangalore, India, Jan 03-06.
- [5] Khalid Sayood, "An Introduction to Data Compression", Academic Press, 1996.
- [6] David Solomon, "Data Compression: The Complete Reference", Springer Publication, 2000.
- [7] M. Atallah and Y. Genin, "Pattern matching text compression: Algorithmic and empirical results", International Conference on Data Compression, vol II: pp. 349-352, Lausanne, 1996.
- [8] Mark Nelson and Jean-Loup Gailly, "The Data Compression Book", Second Edition, M&T Books.
- [9] Timothy C. Bell, "Text Compression", Prentice Hall Publishers, 1990.
- [10] Ranjan Parekh, "Principles of Multimedia", Tata McGraw-Hill Companies, 2006.
- [11] Amiya Halder, Sourav Dey, Soumyadeep Mukherjee and Ayan Banerjee, "An Efficient Image Compression Algorithm Based on Block Optimization and Byte Compression", ICISA-2010, Chennai, Tamilnadu, India, pp. 14-18, Feb 6, 2010.
- [12] Ayan Banerjee and Amiya Halder, "An Efficient Image Compression Algorithm Based on Block Optimization, Byte Compression and Run-Length Encoding along Y-axis", IEEE ICCSIT 2010, Chengdu, China, IEEE Computer Society Press, July 9-11, 2010.
- [13] Debashis Chakraborty, Sandipan Bera, Anil Kumar Gupta and Soujit Mondal, "Efficient Data Compression using Character Replacement through Generated Code", IEEE NCETACS 2011, Shillong, India, March 4-5, 2011, pp 31-34.